Chapitre 18

Attribution d'une adresse à une carte réseau

Après avoir détecter une carte réseau, la seconde étape consiste à lui attribuer une adresse. Il s'agit d'une adresse IP dans le cas d'Internet et pour presque tous les réseaux actuels.

Nous avons vu au chapitre 15 comment les pilotes de périphériques sont enregistrés. Une fois un périphérique réseau enregistré, avant que le noyau puisse gérer les trames (réception et émission), il doit l'activer et lui assigner une adresse. De même le noyau le désactivera lorsqu'il n'en a plus besoin, ce qui est important pour l'utilisateur dans le cas d'une liaison par téléphone. Comme pour les autres périphériques, on parle également sous Linux d'*ouverture* et de *fermeture*. Nous allons voir dans ce chapitre comment ceci est réalisé.

Le noyau n'a aucune raison d'ouvrir ou de fermer le périphérique par lui-même. C'est l'utilisateur qui décide du moment pour cela, heureusement d'ailleurs quand il s'agit d'une connexion téléphonique sur une ligne très onéreuse. Unix BSD a prévu une commande, appelée `ifconfig`, inclue dans les utilitaires `net-tools`, pour ce faire. Cette commande est souvent appelée par un script, aussi est-elle cachée à l'utilisateur lambda.

Nous allons commencer par dire ce qui est nécessaire de cette commande (dont le code ne fait pas parti du noyau Linux) pour l'ouverture et la fermeture d'un périphérique réseau. Celle-ci sert à bien d'autres propos.

18.1 La commande `ifconfig`

18.1.1 Fonctionnalité

La commande `ifconfig` (pour *InterFace CONFIGuration*) n'est pas seulement nécessaire pour l'activation et la désactivation des périphériques réseau, mais également pour leur configuration. Elle permet de définir les paramètres spécifiques aux protocoles, les adresses et les masques de sous-réseau ainsi que de modifier les paramètres d'interface de la carte réseau (paramètres matériels tels que ports d'entrée-sortie, interruptions, etc.). Elle peut également modifier les drapeaux (`ARP`, `PROMISC`, etc.) d'un périphérique réseau. Le périphérique doit être enregistré avant que l'on puisse utiliser cette commande à son égard.

Notre but n'est pas ici d'étudier cette commande dans son intégralité, mais uniquement la partie qui concerne l'activation et la désactivation d'un périphérique réseau. Pour une étude complète, on pourra se référer à la page `man` concernant `ifconfig`.

18.1.2 Liste des périphériques réseau en activité

À l'appel de cette commande sans paramètre, toutes les interfaces qui ont été précédemment ouvertes sont listées l'une après l'autre avec un certain nombre de renseignements :

```
linux:~ # ifconfig
eth0      Link encap:Ethernet  HWaddr 00:04:75:EA:1C:7B
          inet addr:192.168.5.10  Bcast:192.168.5.255  Mask:255.255.255.0
          inet6 addr: fe80::204:75ff:feea:1c7b/10 Scope:Link
          UP BROADCAST NOTRAILERS RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:528 errors:0 dropped:0 overruns:0 frame:0
          TX packets:495 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:487357 (475.9 Kb)  TX bytes:127052 (124.0 Kb)
          Interrupt:5 Base address:0xb000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:30 errors:0 dropped:0 overruns:0 frame:0
          TX packets:30 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1956 (1.9 Kb)  TX bytes:1956 (1.9 Kb)

linux:~ #
```

Dans le cas que nous prenons en exemple, une carte Ethernet et le périphérique en boucle sont activés. Les paramètres en sont les suivants :

- Les noms sont `eth0` et `lo`, ce que l'on pouvait prévoir.
- Les types matériels sont `Ethernet` et `Local Loopback`.
- L'adresse MAC de la carte Ethernet est `00:04:75:ea:1c:7b`.
- L'adresse Internet qui a été affectée à la carte (par un script, en fait ici par DHCP) est `192.168.5.10`. Celle du périphérique en boucle est `127.0.0.1`, par tradition.
- L'adresse de diffusion générale de la carte Ethernet est `192.168.5.255`, ce que l'on peut déduire du masque.
- Le masque de sous-réseau est `255.255.255.0` pour la carte Ethernet et `255.0.0.0` pour le périphérique en boucle, ce qui correspond à des valeurs classiques.

- L'adresse IPv6 de la carte Ethernet est `fe80::204:75ff:feea:1c7b/10` et celle du périphérique en boucle `::1/128`.
- Les portées sont *Link* pour la carte Ethernet et *Host* pour le périphérique en boucle.
- On comprend la valeur des drapeaux : `UP` puisque les interfaces sont activées, `BROADCAST` lorsqu'il y a possibilité de diffusion générale, `MULTICAST` lorsqu'il y a possibilité de multidiffusion et ainsi de suite.
- L'unité maximum de transfert (MTU) est de 1 500 octets pour la carte Ethernet et 16 436 octets pour le périphérique en boucle, ce qui est classique.
- La **métrique** est de 1, c'est-à-dire que cette voie de communication n'est pas encombrée. Une métrique élevée indique que la voie de communication qui part de la carte réseau n'est pas très bonne.
- L'interface Eternet a reçu 528 paquets (RX), sans aucune erreur survenue.
- Elle a transmis 495 (TX), également sans aucune erreur survenue.
- Aucune collision n'est survenue pour la carte Ethernet. La longueur de sa file d'attente en émission est de 100 trames, la valeur par défaut comme nous l'avons vu.
- On peut consulter les nombres d'octets transmis et reçus.
- On a le numéro d'interruption et l'adresse de base d'entrée-sortie de la carte Ethernet.

18.1.3 Activation des périphériques du noyau

L'option `-a` permet d'activer les périphériques réseau qui n'ont pas encore été activés mais qui sont reconnus dans le noyau (pas dans les modules) :

```
linux:~ # ifconfig -a
```

18.1.4 Activation et désactivation d'un périphérique modularisé

L'activation d'un périphérique réseau s'effectue avec la commande :

`ifconfig` *name address* `up`

où *nom* est le nom de l'interface (par exemple `eth0`) et *address* l'adresse à lui attribuer.

Pour désactiver le périphérique réseau, on utilise la commande :

`ifconfig` *name* `down`

Nous n'allons pas étudier ici le lien avec les fonctions des deux sections suivantes : ceci est l'objet d'un programme utilisateur. Toujours est-il que les commandes d'activation et de désactivation finissent par faire appel à ces fonctions.

18.2 Aspect général de l'activation

Nous allons étudier la fonction d'ouverture d'un périphérique réseau, qui renvoie à une fonction spécifique à la carte réseau, puis la fonction spécifique dans le cas de la carte 3Com 501.

18.2.1 Fonction d'ouverture

L'activation d'un périphérique réseau s'effectue par l'intermédiaire de la fonction `dev_open()`. Elle est définie dans le fichier `linux/net/core/dev.c` :

Code Linux 2.6.10

```
808 /**
809  *      dev_open        - prepare une interface pour son utilisation.
810  *      @dev :   peripherique a ouvrir
```

```
811  *
812  *      Passe un peripherique de l'etat desactive a l'etat active. La fonction
813  *      privee du peripherique est appelee et les listes de multidiffusion sont chargees.
814  *      Puis le peripherique est amene dans l'etat actif et un message %NETDEV_UP est
815  *      envoye a la chaine de notification netdev.
816  *
817  *      L'appel de cette fonction pour une interface activee est une nop. En cas d'echec
818  *      un code errno negatif est renvoye.
819  */
820 int dev_open(struct net_device *dev)
821 {
822         int ret = 0;
823
824         /*
825          *      Est-elle deja activee ?
826          */
827
828         if (dev->flags & IFF_UP)
829                 return 0;
830
831         /*
832          *      Est-elle deja presente ?
833          */
834         if (!netif_device_present(dev))
835                 return -ENODEV;
836
837         /*
838          *      Appelle la methode open privee du peripherique
839          */
840         set_bit(__LINK_STATE_START, &dev->state);
841         if (dev->open) {
842                 ret = dev->open(dev);
843                 if (ret)
844                         clear_bit(__LINK_STATE_START, &dev->state);
845         }
846
847         /*
848          *      Si l'ouverture est OK alors :
849          */
850
851         if (!ret) {
852                 /*
853                  *      Positionner les drapeaux.
854                  */
855                 dev->flags |= IFF_UP;
856
857                 /*
858                  *      Initialiser le statut de multidiffusion
859                  */
860                 dev_mc_upload(dev);
861
862                 /*
863                  *      Reveiller l'engin de file d'attente en transmission
864                  */
865                 dev_activate(dev);
866
867                 /*
868                  *      ... et annoncer la nouvelle interface.
869                  */
870                 notifier_call_chain(&netdev_chain, NETDEV_UP, dev);
871         }
872         return ret;
```

```
873 }
```

Autrement dit :

- Si le périphérique passé en argument est déjà ouvert, on renvoie 0.
- Si le périphérique n'est pas présent, on renvoie l'opposé du code d'erreur `ENODEV`.

  La fonction `netif_device_present()` est définie dans le fichier `linux/include/linux/netdevice.h`: Code Linux 2.6.10

```
72608 /* Chargement a chaud. */
727 static inline int netif_device_present(struct net_device *dev)
728 {
729         return test_bit(__LINK_STATE_PRESENT, &dev->state);
730 }
```

- On spécifie comme champ d'état du descripteur de périphérique que ce dernier est en train de démarrer.
- S'il existe une fonction spécifique d'ouverture correspondant à la carte réseau, on fait appel à elle. Si tout ne s'est pas bien déroulé, on met à zéro le bit `__LINK_STATE_START` du champ d'état du descripteur de périphérique.

  Nous étudierons ci-dessous le cas de la fonction spécifique d'ouverture pour la carte 3Com 501.
- Si tout s'est bien passé, on positionne le drapeau `IFF_UP`, on initialise le statut de multidiffusion, qui ne nous intéressera pas dans l'immédiat, on active la file d'attente en émission et l'ordonnanceur du périphérique (par appel à la fonction `dev_activate()` étudiée ci-dessous) et on annonce la nouvelle interface.

18.2.2 Activation de la file d'attente en émission

La fonction `dev_activate()` est définie dans le fichier `linux/net/sched/sch_generic.c`: Code Linux 2.6.10

```
515 void dev_activate(struct net_device *dev)
516 {
517         /* Aucune strategie de mise en file d'attente n'est attachee au peripherique ;
518            en creer une par defaut, a savoir pfifo_fast pour les peripheriques,
519            ce qui necessite une mise en file d'attente, et noqueue_qdisc pour les
520            interfaces virtuelles
521          */
522
523         if (dev->qdisc_sleeping == &noop_qdisc) {
524                 struct Qdisc *qdisc;
525                 if (dev->tx_queue_len) {
526                         qdisc = qdisc_create_dflt(dev, &pfifo_fast_ops);
527                         if (qdisc == NULL) {
528                                 printk(KERN_INFO "%s: activation failed\n", dev->name);
529                                 return;
530                         }
531                         write_lock_bh(&qdisc_tree_lock);
532                         list_add_tail(&qdisc->list, &dev->qdisc_list);
533                         write_unlock_bh(&qdisc_tree_lock);
534                 } else {
535                         qdisc =  &noqueue_qdisc;
536                 }
537                 write_lock_bh(&qdisc_tree_lock);
538                 dev->qdisc_sleeping = qdisc;
539                 write_unlock_bh(&qdisc_tree_lock);
540         }
541
```

```
542         spin_lock_bh(&dev->queue_lock);
543         rcu_assign_pointer(dev->qdisc, dev->qdisc_sleeping);
544         if (dev->qdisc != &noqueue_qdisc) {
545                 dev->trans_start = jiffies;
546                 dev_watchdog_up(dev);
547         }
548         spin_unlock_bh(&dev->queue_lock);
549 }
```

Autrement dit :

- Si le champ `qdisc_sleeping` du descripteur de périphérique passé en argument est `noop_qdisc` (stratégie de mise en file d'attente qui ne fait rien, placée lors de l'installation du pilote de périphérique comme nous l'avons vu) :
  - Si la longueur de la file d'attente en émission est non nulle : on essaie d'instantier une stratégie de file d'attente par défaut ; si on n'y parvient pas, on affiche un message noyau et on a terminé ; sinon on ajoute cette stratégie à la liste des stratégies du descripteur de périphérique.

    La fonction `qdisc_create_dflt()` et `pfifo_fast_ops` sont définis dans le fichier `linux/net/sched/sch_generic.c` :

Code Linux 2.6.10

```
399 static struct Qdisc_ops pfifo_fast_ops = {
400         .next           =       NULL,
401         .cl_ops         =       NULL,
402         .id             =       "pfifo_fast",
403         .priv_size      =       3 * sizeof(struct sk_buff_head),
404         .enqueue        =       pfifo_fast_enqueue,
405         .dequeue        =       pfifo_fast_dequeue,
406         .requeue        =       pfifo_fast_requeue,
407         .init           =       pfifo_fast_init,
408         .reset          =       pfifo_fast_reset,
409         .dump           =       pfifo_fast_dump,
410         .owner          =       THIS_MODULE,
411 };
412
413 struct Qdisc * qdisc_create_dflt(struct net_device *dev, struct Qdisc_ops *ops)
414 {
415         void *p;
416         struct Qdisc *sch;
417         int size;
418
419         /* S'assurer que le Qdisc et les donnees privees sont alignees sur 32
               octets */
420         size = ((sizeof(*sch) + QDISC_ALIGN_CONST) & ~QDISC_ALIGN_CONST);
421         size += ops->priv_size + QDISC_ALIGN_CONST;
422
423         p = kmalloc(size, GFP_KERNEL);
424         if (!p)
425                 return NULL;
426         memset(p, 0, size);
427
428         sch = (struct Qdisc *)(((unsigned long)p + QDISC_ALIGN_CONST)
429                                & ~QDISC_ALIGN_CONST);
430         sch->padded = (char *)sch - (char *)p;
431
432         INIT_LIST_HEAD(&sch->list);
433         skb_queue_head_init(&sch->q);
434         sch->ops = ops;
435         sch->enqueue = ops->enqueue;
436         sch->dequeue = ops->dequeue;
```

```
437         sch->dev = dev;
438         dev_hold(dev);
439         sch->stats_lock = &dev->queue_lock;
440         atomic_set(&sch->refcnt, 1);
441         if (!ops->init || ops->init(sch, NULL) == 0)
442                 return sch;
443
444         dev_put(dev);
445         kfree(p);
446         return NULL;
447 }
```

La constante `QDISC_ALIGN_CONST` est définie dans le fichier `linux/include/net/pkt_sched.h` : Code Linux 2.6.10

```
16 #define QDISC_ALIGN             32
17 #define QDISC_ALIGN_CONST       (QDISC_ALIGN - 1)
```

– On renseigne le champ de stratégie assoupie du descripteur de périphérique, soit avec `noqueue_qdisc`, soit avec celle qui vient d'être instantiée.

La stratégie `noqueue_qdisc` est définie dans le fichier `linux/net/sched/sch_generic.c` : Code Linux 2.6.10

```
286 struct Qdisc_ops noqueue_qdisc_ops = {
287         .next           =       NULL,
288         .cl_ops         =       NULL,
289         .id             =       "noqueue",
290         .priv_size      =       0,
291         .enqueue        =       noop_enqueue,
292         .dequeue        =       noop_dequeue,
293         .requeue        =       noop_requeue,
294         .owner          =       THIS_MODULE,
295 };
296
297 struct Qdisc noqueue_qdisc = {
298         .enqueue        =       NULL,
299         .dequeue        =       noop_dequeue,
300         .flags          =       TCQ_F_BUILTIN,
301         .ops            =       &noqueue_qdisc_ops,
302         .list           =       LIST_HEAD_INIT(noqueue_qdisc.list),
303 };
```

– On verrouille la file d'attente du descripteur de périphérique.
– On renseigne le champ `qdisc` de celui-ci avec la stratégie en attente de ce même descripteur.
– Si celle-ci est différente de `noqueue_disc`, on renseigne l'heure de la dernière émission avec l'heure en cours et on démarre le minuteur permettant l'identification de problèmes en émission.

La fonction `dev_watchdog_up()` sera étudiée ci-dessous.

– On déverrouille la file d'attente du descripteur de périphérique.

18.2.3 Démarrage du minuteur de problèmes d'émission

La méthode `dev_watchdog_up()` démarre le minuteur permettant l'identification de problèmes d'émission. Elle est définie dans le fichier `linux/net/sched/sch_generic.c` : Code Linux 2.6.10

```
215 void __netdev_watchdog_up(struct net_device *dev)
216 {
217         if (dev->tx_timeout) {
218                 if (dev->watchdog_timeo <= 0)
```

```
219                         dev->watchdog_timeo = 5*HZ;
220                 if (!mod_timer(&dev->watchdog_timer, jiffies + dev->watchdog_timeo))
221                         dev_hold(dev);
222         }
223 }
224
225 static void dev_watchdog_up(struct net_device *dev)
226 {
227         spin_lock_bh(&dev->xmit_lock);
228         __netdev_watchdog_up(dev);
229         spin_unlock_bh(&dev->xmit_lock);
230 }
```

18.3 Cas de la carte 3Com 501

18.3.1 Installation du gestionnaire d'interruption

Code Linux 2.6.10

Nous avons vu au chapitre 17 que la fonction d'ouverture spécifique à la carte réseau Ethernet 3Com 501 s'appelle `el_open()`. Elle est définie dans le fichier `linux/drivers/net/3c501.c`:

```
327 /**
328  *      el1_open :
329  *      @dev : peripherique qui est en train d'etre ouvert
330  *
331  *      Lorsqu'un ifconfig change les drapeaux du peripherique pour inclure
332  *      IFF_UP, cette fonction est appelee. Elle est seulement appelee lorsque ce changement
333  *      apparait, pas lorsque l'interface reste activee. #el1_close sera appelee
334  *      lorsqu'il est desactive.
335  *
336  *      Renvoie 0 en cas de succes d'ouverture, ou -EAGAIN si quelqu'un a tire
337  *      notre ligne d'interruption.
338  */
339
340 static int el_open(struct net_device *dev)
341 {
342         int retval;
343         int ioaddr = dev->base_addr;
344         struct net_local *lp = netdev_priv(dev);
345         unsigned long flags;
346
347         if (el_debug > 2)
348                 printk(KERN_DEBUG "%s: Doing el_open()...", dev->name);
349
350         if ((retval = request_irq(dev->irq, &el_interrupt, 0, dev->name, dev)))
351                 return retval;
352
353         spin_lock_irqsave(&lp->lock, flags);
354         el_reset(dev);
355         spin_unlock_irqrestore(&lp->lock, flags);
356
357         lp->txing = 0;          /* Carte en mode RX */
358         outb(AX_RX, AX_CMD);    /* Controle Aux : irq et reception actives */
359         netif_start_queue(dev);
360         return 0;
361 }
```

Autrement dit :

- On déclare une valeur de retour.
- On déclare un numéro de port d'entrée-sortie, que l'on initialise avec celui fourni par le descripteur de périphérique passé en argument.

- On déclare une structure locale de réseau, que l'on initialise avec la partie privée du descripteur de périphérique passé en argument.
- On déclare un vecteur de drapeaux.
- Si le niveau de débogage est strictement supérieur à 2, on affiche un message noyau indiquant qu'on est en train d'ouvrir le périphérique.
- On essaie de mettre en place le gestionnaire d'interruption `el_interrupt()` correspondant au numéro d'interruption de la carte (celui-ci étant fourni par le descripteur de périphérique passé en argument). Si une erreur se produit, on renvoie le code d'erreur fourni par la fonction `request_irq()`.

  Le gestionnaire d'interruption sera étudié au chapitre 19.
- On sauve les valeurs des registres dans le vecteur des drapeaux.
- On (ré-)initialise le périphérique, grâce à la fonction `el_reset()` étudiée ci-après.
- On restaure les valeurs des registres.
- On se place en mode de réception et on renvoie 0.

  La fonction `netif_start_queue()` est définie dans le fichier `linux/include/linux/-netdevice.h`: Code Linux 2.6.10

```
606 static inline void netif_start_queue(struct net_device *dev)
607 {
608         clear_bit(__LINK_STATE_XOFF, &dev->state);
609 }
```

18.3.2 Réinitialisation

La fonction `el_reset()` de réinitialisation spécifique à la carte 3Com 501 est définie dans le fichier `linux/drivers/net/3c501.c`: Code Linux 2.6.10

```
756 /**
757  * el_reset : Reinitialise une carte 3c501
758  * @dev : La carte 3c501 a traiter
759  *
760  * Meme la reinitialisation d'un 3c501 n'est pas simple. Lorsque vous activez la
761  * reinitialisation, elle perd tout de sa configuration. Vous devez detenir le verrou
       lorsque vous faites ca. La fonction
762  * ne peut pas tirer le verrou elle-meme car elle est appelable depuis la routine d'irq.
763  */
764
765 static void  el_reset(struct net_device *dev)
766 {
767         struct net_local *lp = netdev_priv(dev);
768         int ioaddr = dev->base_addr;
769
770         if (el_debug> 2)
771                 printk(KERN_INFO "3c501 reset...");
772         outb(AX_RESET, AX_CMD);         /* Reinitialise la puce */
773         outb(AX_LOOP, AX_CMD);          /* Controle aux : irq et loopback actives */
774         {
775                 int i;
776                 for (i = 0; i < 6; i++) /* Positionne l'adresse de la station. */
777                         outb(dev->dev_addr[i], ioaddr + i);
778         }
779
780         outw(0, RX_BUF_CLR);            /* Initialise la zone de rx des paquets a 0. */
781         outb(TX_NORM, TX_CMD);          /* Active l'irq en tx, collision */
782         outb(RX_NORM, RX_CMD);          /* Initialise les commandes de Rx. */
783         inb(RX_STATUS);                 /* Efface le statut. */
784         inb(TX_STATUS);
```

```
785         lp->txing = 0;
786 }
```

Autrement dit :

- on déclare une structure locale de réseau, que l'on initialise avec la partie privée du descripteur de périphérique passé en argument ;
- on déclare un port d'entrée-sortie, que l'on initialise avec celui fourni par le descripteur de périphérique passé en argument ;
- si le niveau de débogage est strictement supérieur à 2, on affiche un message noyau ;
- on passe l'ordre de réinitialisation à la carte du périphérique, spécifique à la carte 3Com 501 ;
- on active les fonctions de la carte (IRQ et périphérique en boucle) ;
- on positionne l'adresse de la station sur la carte ;
- on initialise le pointeur du tampon de réception à zéro ;
- on indique qu'il n'y a pas de traitement en transmission à effectuer ;
- on envoie la commande de réception ;
- on met à zéro le statut ;
- on se place en mode de réception.